ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
(РОСАВИАЦИЯ)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра прикладной математики

А.А. Егорова, Н.Л. Рогожникова

АЛГОРИТМЫ ДИСКРЕТНОЙ МАТЕМАТИКИ

Учебно-методическое пособие
по выполнению контрольного домашнего задания

для студентов I курса
специальности 01.03.04
очной формы обучения

Москва
ИД Академии Жуковского
2021

УДК 519.6
ББК 517
Е30

Рецензент:
Овсянникова Н.И. – канд. физ.-мат. наук, доцент

Егорова А.А.
Е30 Алгоритмы дискретной математики [Текст] : учебно-методическое пособие по выполнению контрольного домашнего задания / А.А. Егорова, Н.Л. Рогожникова. – М.: ИД Академии Жуковского, 2021. – 48 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Алгоритмы дискретной математики» по учебному плану для студентов I курса специальности 01.03.04 очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 20.05.2021 г. и методического совета 20.05.2021 г.

УДК 519.6
ББК 517

В авторской редакции

Подписано в печать 22.09.2021 г.
Формат 60x84/16 Печ. л. 3 Усл. печ. л. 2,79
Заказ № 801/0616-УМП14 Тираж 25 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (495) 973-45-68
E-mail: zakaz@itsbook.ru

Оглавление

1. ВВЕДЕНИЕ

Настоящее учебное пособие предназначено для студентов первого курса специальности 01.03.04, изучающих дисциплину «Алгоритмы дискретной математики». Дисциплина относится к обязательным дисциплинам базовой части образовательной программы направления подготовки 01.03.04, квалификация бакалавр.

Пособие предназначено для выполнения контрольного домашнего задания (КДЗ), содержит задание на выполнение и индивидуальные варианты для самостоятельной работы каждого студента. Краткие теоретические сведения разделены на 3 раздела в соответствии с темами задания, в каждом разделе приведены методы решения задач и соответствующие примеры.

1.1 Порядок выполнения, сдача и защита КДЗ

КДЗ выдается после прохождения тем по дисциплине, включенных в КДЗ, и выполняется студентом самостоятельно вне учебных аудиторий в точном соответствии с заданием и вариантом по списку. При необходимости преподавателю можно задать вопросы во время практического занятия или лабораторной работы.

В процессе подготовки к выполнению КДЗ студент:

- изучает лекционный материал, материалы данного пособия и дополнительной литературы по темам контрольного домашнего задания;
- знакомится с заданием (при необходимости задает преподавателю уточняющие вопросы);
- готовит отчет по выполнению КДЗ (пункты, отмеченные знаком *).

В срок, установленный преподавателем, не позднее, чем за 3 дня до предпоследнего в семестре практического занятия, студент сдает отчет по КДЗ, который должен содержать:

- название работы*;
- задание на выполнение КДЗ*;
- варианты по каждому пункту задания*;

- решения с комментариями.

На предпоследнем практическом занятии осуществляется защита КДЗ преподавателю (ответы на вопросы, исправление ошибок, переделка заданий в случае необходимости и т.п.)

После защиты КДЗ студентом преподавателем делается соответствующая запись на отчете студента, который сдается для хранения на кафедру. Студенты, получившие за КДЗ оценку «неудовлетворительно» или не сдавшие отчет по КДЗ в срок, к экзамену по дисциплине Алгоритмы дискретной математики не допускаются.

1.2 Требования к оформлению КДЗ

КДЗ оформляется либо в тетради, либо на листах формата А4, «вшитых» в папку или жестко скрепленных иным способом. Оформление может быть рукописным (ручка черная или синяя) или на персональном компьютере.

Студент последовательно выполняет задания, записывая решения в отчет.

На титульном листе указывается фамилия студента, группа, дисциплина, по которой выполняется КДЗ, номер варианта, фамилия и инициалы преподавателя.

В тетради (на листах) записывается задание, данные в соответствии со своим вариантом и далее подробно – решение (со всеми промежуточными результатами). Ответ записывается отдельной строкой или выделяется иным понятным преподавателю способом.

Решение должно быть оформлено аккуратно, без зачеркивания и многократных «замазываний». Рукописный текст не должен «падать» со строки, цифры и буквы должны быть написаны разборчиво.

1.3. Задание на выполнение КДЗ

Каждый вариант КДЗ включает в себя следующий набор из восьми задач:

1. Построить нормальный алгоритм Маркова.

2. Построить машину Тьюринга.

3. Разработать, реализовать на языке Pascal (Python) и оценить вычислительную сложность итеративного алгоритма сортировки.

4. Пронумеровать строки в программе (фрагменте программы), реализующей комбинаторный алгоритм, и подсчитать количество выполнений указанных в задании строк.

5. Найти решение рекуррентного соотношения.

6. Найти последовательность, соответствующую заданной производящей функции.

7. Найти количество способов представления заданного числа в виде суммы определенных чисел, используя представление последовательности в виде суммы ряда,

8. Оценить сумму ряда предложенным методом.

Индивидуальные задания в соответствии с номером варианта представлены в конце соответствующих разделов пособия.

2. СТАНДАРТИЗАЦИЯ АЛГОРИТМОВ

2.1. Краткие теоретические сведения

Первые стандартизованные варианты понятия алгоритм разработаны в 30-х годах XX века в работах А. Тьюринга, А. Чёрча и Э. Поста. Предложенные ими машина Тьюринга, машина Поста и лямбда-исчисление Чёрча оказались эквивалентными друг другу.

Алгоритм (algorithm) – любая корректно определенная вычислительная процедура, на вход (input) которой подается некоторая величина или набор величин **(аргумент алгоритма)**, и результатом выполнения которой является выходная (**output**) величина или набор значений (**результат**).

Правильный (корректный) алгоритм - такой алгоритм, который после каждого ввода данных, своим результатом имеет корректный **вывод**. Корректный алгоритм решает данную вычислительную задачу.

2.1.1. Нормальный алгоритм Маркова

Норма́льный алгори́тм Ма́ркова (**НАМ** или **марковский алгоритм**) - один из стандартных способов формального определения понятия алгоритма Нормальные алгоритмы являются вербальными, то есть представляют собой правила по переработке слов в некоторых алфавитах.

Определение всякого нормального алгоритма состоит из двух частей: определения алфавита алгоритма и определения его схемы.

Схемой нормального алгоритма называется конечный упорядоченный набор формул подстановки, каждая из которых может быть простой или заключительной.

Простыми формулами подстановки называются слова вида $P \to Q$, где P и Q — два произвольных слова в алфавите алгоритма (называемые, соответственно, **левой и правой частями формулы подстановки**).

Заключительными формулами подстановки называются слова вида $P \to \cdot Q$, где P и Q — два произвольных слова в алфавите алгоритма.

Важно, что вспомогательные буквы $\to$ и $\to \cdot$ не принадлежат алфавиту алгоритма (в противном случае на исполняемую ими роль разделителя левой и правой частей следует избрать другие две буквы).

Принцип нормализации Маркова: всякий алгоритм может быть реализован нормальным алгоритмом Маркова (или всякий алгоритм нормализуем).

2.1.2. Машина Тьюринга

Другим известным способом задания алгоритма является машина Тьюринга (МТ).

Машина Тьюринга состоит из:

1) управляющего устройства, которое может находиться в одном из состояний, образующих конечное множество $Q = \{q_1, \ldots, q_n\}$;

2) бесконечной ленты, разбитой на ячейки, в каждой из которых может быть задан один из символов конечного алфавита $A = \{a_1, \ldots, a_m\}$;

3) устройства обращения к ленте, т.е. считывающей и пишущей головки, которая в каждый момент времени обозревает ячейку ленты и в зависимости от символа в этой ячейке и состояния управляющего устройства:

а) записывает в ячейку символ (быть может, совпадающий с прежним или пустой);

б) сдвигается на ячейку влево или вправо, или остается на месте;

в) переходит в новое состояние (или остается в прежнем).

Среди состояний управляющего устройства выделяют начальное состояние q_1 и заключительное состояние q_z.

Стандартной начальной конфигурацией называется конфигурация вида $q_1\alpha$, т.е. конфигурацию содержащую начальное состояние, в котором головка обозревает крайний левый символ слова, написанного на ленте.

Стандартной заключительной конфигурацией называется конфигурация вида $q_z\alpha$.

Данные МТ - это слова в алфавите ленты.

Память МТ - это конечное множество состояний (внутренняя память) и лента (внешняя память). Лента бесконечна в обе стороны. В начальный момент времени только конечное число ячеек ленты заполнено непустыми символами, остальные ячейки пусты, т. е содержат пустой символ λ (пробел).

Машина Тьюринга называется **детерминированной**, если каждой комбинации состояния и ленточного символа в таблице соответствует не более одного правила. Конкретная машина Тьюринга задаётся перечислением элементов множества букв алфавита A, множества состояний Q и набора правил, по которым работает машина.

Правила имеют вид: $q_i a_j \rightarrow q_i' a_j' d_k$.

Если головка находится в состоянии q_i, а в текущей ячейке записана буква a_j, то головка переходит в состояние q_1', в ячейку вместо a_j записывается a_j', головка делает движение d_k, которое имеет три варианта: на ячейку влево

(L), на ячейку вправо (R), остаться на месте (E). Для каждой возможной конфигурации $<q_i, a_j>$ имеется ровно одно правило (для недетерминированной машины Тьюринга может быть большее количество правил.

Элементарные шаги МТ - это считывание и запись символов, сдвиг на ячейку влево или вправо, а также переход управляющего устройства в следующее состояние.

Результатом работы МТ является слово на ленте после остановки машины.

Возможно построение композиций машин Тьюринга. Например, на рисунке 2.1 представлена блок-схема композиций машин T_1 и T_2, которая обозначается $T_2(T_1)$.

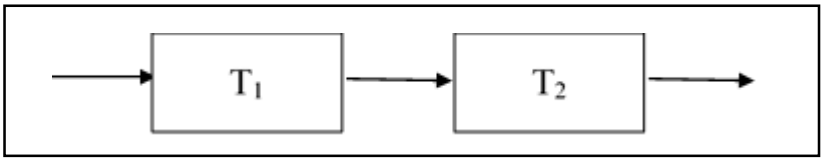


Рисунок 2.1 – Блок-схема композиции $T_2(T_1)$.

Эта блок-схема всегда предполагает, что исходными данными машины T_2 являются результаты T_1. Благодаря вычислимости композиции машин заключительная конфигурация T_2 превращается в стандартную начальную конфигурацию T_1.

Говорят, что машина Тьюринга вычисляет предикат $P(\alpha)$ (α - слово в алфавите A^*), если $T(\alpha) = \omega$, где $\omega = T$, когда $P(\alpha)$ истинно, и $\omega = F$, когда $P(\alpha)$ ложно. Если $P(\alpha)$ не определен, то машина Тьюринга не останавливается.

Говорят, что МТ T вычисляет $P(\alpha)$ **с восстановлением**, если $P(\alpha) = \omega\alpha$.

Вычисление с восстановлением можно представить следующей последовательностью конфигураций:

$$q_1\alpha \Rightarrow q_{n1}\,\alpha\,{*}\alpha \Rightarrow \alpha\,{*}\,q_{n2}\,\alpha \Rightarrow \alpha{*}q_{n3}\,\omega \Rightarrow q_{n4}\,\omega\alpha.$$

Первая часть этой последовательности реализуется машиной T_k (смотри пример 2.2.2); вторая - простым сдвигом головки до маркера «*»; третья – машиной T_R, вычисляющей $P(\alpha)$ на правой полуленте (одного копирования мало для восстановления, так как копию может испортить основная машина T, нужна машина, не заходящая влево от α); четвертая - переносом ω в крайнее

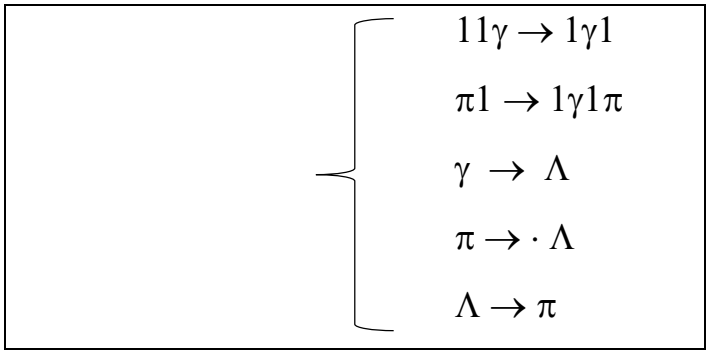
левое положение. Машина T является композицией указанных четырех машин.

2.2. Примеры решения заданий

Число представимо в **единичном (унарном) коде**, когда для всех числовых функций существует алфавит $A = \{1, *, \lambda\}$ и число x представляется словом $1\ldots1 = 1^x$, состоящем из x единиц.

Пример 2.2.1 Построить нормальный алгоритм Маркова, увеличивающий в два раза число, написанное в унарном коде.

Решение: При решении будем использовать алфавит $A = \{1, \pi, \gamma\}$. Тогда схема нормального алгоритма будет иметь вид:



Рассмотрим применение этого алгоритма к слову 1111. В результате должно получиться слово 11111111.

$1111 \to \underline{\pi 1}111 \to 1\gamma 1\underline{\pi 1}11 \to 1\gamma\underline{11\gamma}1\pi 11 \to 1\gamma 1\gamma 11\underline{\pi 1}1 \to 1\gamma 1\gamma 1\underline{11\gamma}1\pi 1 \to 1\gamma 1\gamma\underline{11\gamma}11\pi 1 \to 1\gamma 1\gamma 1\gamma 111\underline{\pi 1} \to 1\gamma 1\gamma 1\gamma 1\underline{11\gamma}1\pi \to 1\gamma 1\gamma 1\gamma\underline{11\gamma}11\pi \to 1\gamma 1\gamma 1\gamma\underline{11\gamma}111\pi \to 1\gamma 1\gamma 1\gamma 1\gamma 1111\pi \to \underline{11\gamma}1\gamma 1\gamma 1111\pi \to 1\gamma\underline{11\gamma}1\gamma 1111\pi \to 1\gamma 1\gamma\underline{11\gamma}1111\pi \to 1\gamma 1\gamma 1\gamma 11111\pi \to \underline{11\gamma}1\gamma 11111\pi \to 1\gamma\underline{11\gamma}11111\pi \to 1\gamma 1\gamma 111111\pi \to \underline{11\gamma}111111\pi \to 1\gamma 1111111\pi \to 11111111\underline{\pi} \to 11111111$

Пример 2.2.2 Построить машину Тьюринга T_k, осуществляющую копирование слова.

Решение: Нам нужно переработать слово β в слово $\beta * \beta$. Для чисел задачу копирования решает машина T_k со следующей системой команд:

$q_1* \to q_1*L$ $\quad$ $q_2\lambda \to q_3*R$

$q_1 1 \to q_2 0R$ $\quad$ $q_3 1 \to q_3 1R$

$q_1\lambda \to q_z\lambda R$ $\quad$ $q_3\lambda \to q_4 1L$

$q_1 0 \to q_1 1L$ $\quad$ $q_4 1 \to q_4 1L$

$q_2* \to q_3*R$ $\quad$ $q_4* \to q_4*L$

$q_2 1 \to q_2 1R$ $\quad$ $q_4 0 \to q_4 0R$

При первом проходе в состояние q_2 ставится символ «*» справа от исходного числа 1^β. Далее при каждом проходе исходного числа МТ заменяет левую единицу числа нулем и пишет в состоянии q_3 одну единицу справа от 1^β. Копия 1^β строится за β проходов. После записи очередной единицы машина T_k переходит в состояние q_4, которое передвигает головку влево до ближайшего нуля, после чего T_k переходит в состояние q_1, и цикл повторяется до тех пор, пока не обнаружится маркер «*» - головка возвращается влево в свое исходное положение, заменяя по дороге все нули единицами.

Построим диаграмму переходов (см. рисунок 2.2).

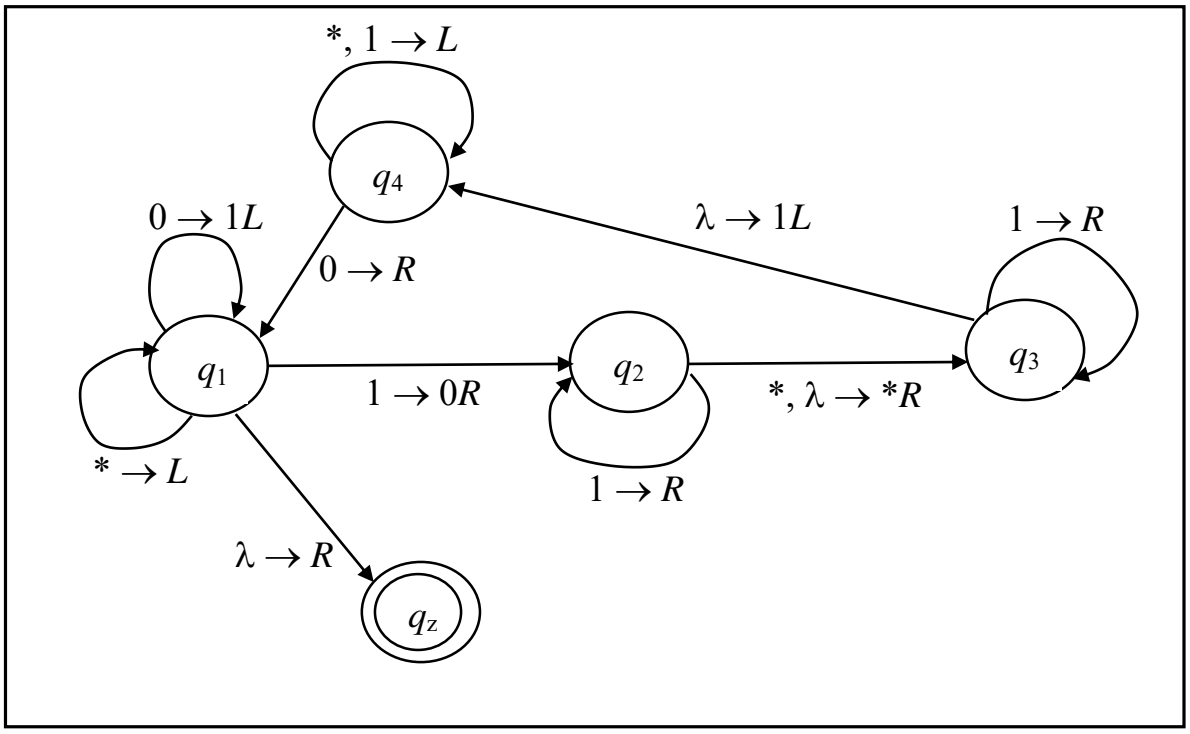


Рисунок 2.2 – Диаграмма переходов МТ для копирования числа в унарном коде. На диаграмме принимаем следующие сокращения:

1) Если из q_i в q_j ведут два ребра с одинаковой правой частью, то они объединяются в одно ребро, на котором левые части записаны через запятую;

2) Если символ на ленте не изменяется, то он в правой части команды не пишется.

2.3. Варианты заданий.

Задание 1.

Номер варианта	Задание на выполнение
1.	Построить НАМ для вычисления функции $f(x) = x - 1$ ($x > 1$) в унарной системе счисления.
2.	Построить НАМ, заменив в некотором исходном слове из алфавита $A = \{a, b, c\}$ все буквы a на c. Используйте символ ε для расширения алфавита A. $B = A \cup \varepsilon$.
3.	Дано слово в алфавите $A = \{a, b, c\}$. Построить НАМ, присоединяющий слово Q к данному слову (используйте маркер).
4.	Постойте НАМ, определяющим делится ли число в унарном коде на 3.
5.	Дан алфавит $A = \{1, 2\}$. Расширив алфавит A двумя буквами $B = A \cup \{\alpha, \beta\}$, постройте НАМ, записывающий слово в обратном порядке.
6.	Построить НАМ, который в слове из алфавита $A = \{a, b, c, d, e, f\}$ удаляет все вхождения последовательности bc.
7.	Определите, что выполняет НАМ, заданный следующей схемой формул подстановки: $\varepsilon a \to a\varepsilon$ $\varepsilon c \to c\varepsilon$ $\Lambda \to \varepsilon$ $\varepsilon b \to b\varepsilon$ $\varepsilon \to \cdot Q$

8.	Дано слово в алфавите $A = \{a, b, c\}$. Построить НАМ, который удаляет все c в некотором слове.
9.	Построить НАМ для вычисления функции $f(x) = x \bmod 3$ в унарной системе счисления.
10.	Построить НАМ, который в слове из алфавита $A = \{a, b, c, d, e, f\}$ все символы d заменяет на e , а e на de.
11.	Построить НАМ, который в любом алфавите $A = \{a, b\}$ переносит все буквы b в начало слова.
12.	Построить НАМ, который проверяет четность числа, записанного в десятичной системе счисления.
13.	Построить НАМ для перевода числа из двоичной системы счисления в четверичную систему счисления.
14.	Построить НАМ, переводящий число из четвертичной системы счисления в двоичную систему счисления.
15.	Построить НАМ для вычисления функции $f(x) = x - 1$ $(x > 1)$ в троичной системе счисления.
16.	Построить НАМ для вычисления функции $f(x) = x + 1$ в десятичной системе счисления.
17.	В процессе работы НАМ получилась следующая последовательность: $badca \to bdca \to bdc \to dc \to d \to \Lambda \to \cdot\Lambda$. Определите возможную схему для данных переводов.
18.	Построить НАМ для перевода числа из восьмеричной системы счисления в двоичную систему счисления.
19.	Дано слово в алфавите $A = \{a, b, c\}$. Построить НАМ, который удаляет все b в некотором слове.
20.	Построить НАМ, который в слове из алфавита $A = \{a, b, c, d, e, f\}$ все символы a заменяет на e , а e на ae.

Задание 2.	
Номер варианта	Задание на выполнение
1.	Дан алфавит A = {1, λ} и состояния { q_1, q_2}. Построить зацикливающуюся МТ и ее диаграмму.
2.	Построить диаграмму переходов МТ для сложения двух заданных натуральных чисел, A = {1, λ}.
3.	Построить МТ для получения следующего числа в унарной системе, то есть вычислить функцию $f(x) = x + 1$, A = {1, λ}.
4.	Дан алфавит A = {1, 0}. Построить машину Тьюринга, удаляющую ноль.
5.	Дано слово *btokt* в алфавите A = {*b*, *k*, *o*, *t*, λ}. Построить систему команд машины Тьюринга T*p*, преобразующую данное слово в слово *btktt*.
6.	Построить МТ, которая правильно вычисляет функцию $f(x) = x + 1$ в десятичной системе счисления.
7.	Построить диаграмму переходов МТ T_{-}, которая вычисляет разность двух натуральных чисел.
8.	Даны два набора единиц, разделенные *. Построить машину Тьюринга, которая выбирала бы больший из этих наборов.
9.	Дан алфавит A = {(,), λ} и внутренние состояния Q = {q_1, q_2, …, q_z}. Построить машину Тьюринга, проверяющую верно ли в исходной последовательности расставлены скобки.
10.	По диаграмме, представленной на рисунке 2.3 построить систему команд МТ T_{n+} для суммирования n натуральных чисел.

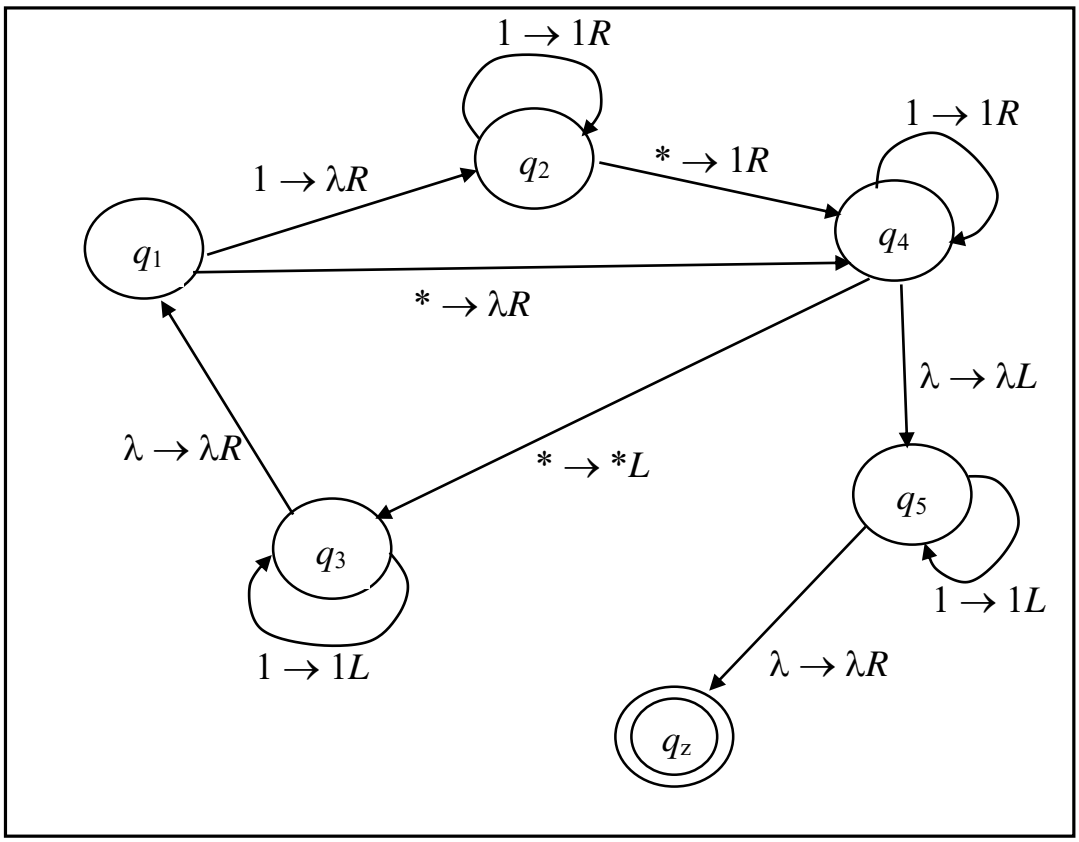


Рисунок 2.3 – Диаграмма переходов машины Тьюринга T_{n+} для сложения *n* чисел.

11.	Построить диаграмму переходов машины T*p* из задания 10.
12.	Для машины Тьюринга с алфавитом $A = \{1, \lambda\}$ и внутренними состояниями $Q = \{ q_1, q_2\}$ со следующей системой команд : $\begin{cases} q_1\lambda \to q_z1 \\ q_11 \to q_11R \end{cases}$ определить, в какое слово переработает машинка каждое из следующих слов, если она находится в начальном состоянии и обозначает указанную ячейку: а) 1λ11λλ11 (обозревается ячейка 3, считая слева) б) 11λ111λ1 (обозревается ячейка 2) в) 1λλ111 (обозревается ячейка 3) Ответ изобразить схематически в виде последовательных конфигураций, возникающих на ленте на каждом такте работы машины.

13. Машина Тьюринга с диаграммой, представленной на рисунке 2.4 вычисляет предикат P(α) «α - четное число»: головка достигает конца числа в состоянии q_2, если число единиц четно, и в состоянии q_3, если число единиц нечетно, после чего она перемещается в исходное положение в состоянии q_4 либо q_5 и печатает T (true) или F (false) соответственно.

Что нужно сделать в петлях q_4 и q_5, чтобы предикат вычислялся с восстановлением, то есть чтобы не происходило уничтожение α?

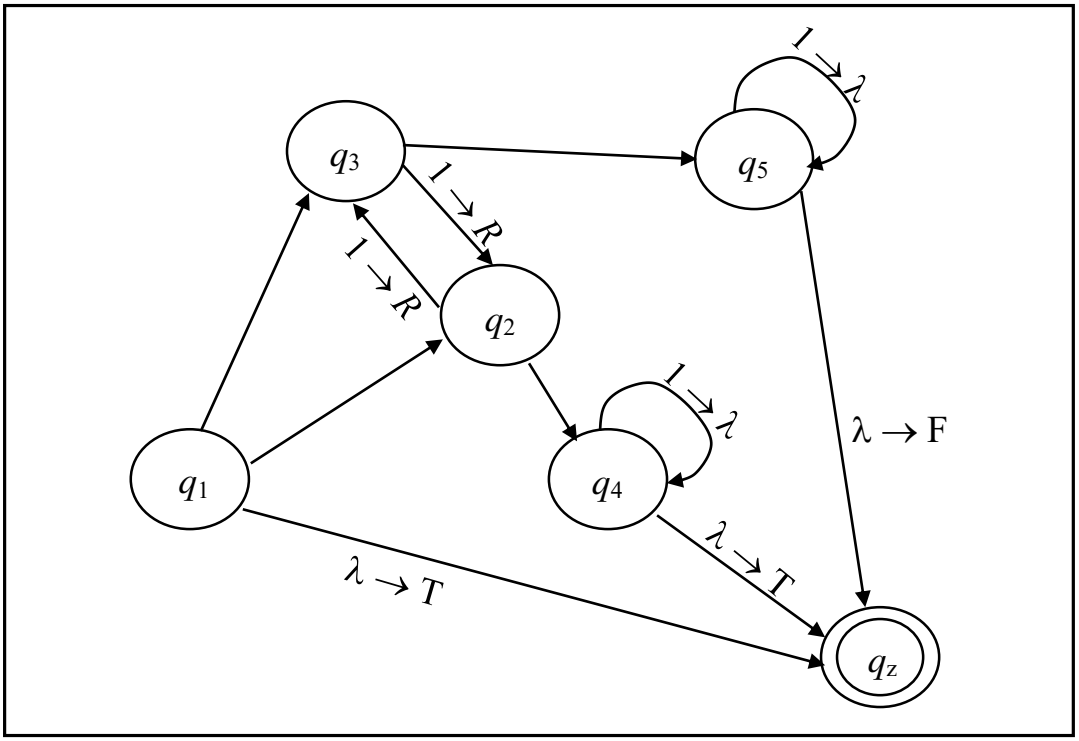


Рисунок 2.4 – Диаграмма переходов машины Тьюринга T_α для вычисления предиката «α - четное число».

14. Для машины Тьюринга с алфавитом $A = \{1, 0, \lambda\}$ и внутренними состояниями $Q = \{ q_1, q_2, q_3\}$ со следующими системами команд:

$a_j \setminus q_i$	q_1	q_2
λ	$q_z 1$	$q_1 \lambda L$
1	$q_2 0 R$	$q_1 1 R$
0	$q_1 1 R$	$q_2 0 L$

15. Построить машину Тьюринга, которая выполняет умножение двух чисел в унарной системе счисления.

16.	Построить диаграмму переходов МТ T_, которая вычисляет разность двух целых чисел.
17.	Даны два набора единиц, разделенные *. Построить машину Тьюринга, которая выбирала бы меньший из этих наборов.
18.	Построить машину Тьюринга, которая вычитает единицу из числа в десятичной системе счисления.
19.	Дан алфавит A = {a, b}. Построить машину Тьюринга, удаляющую b.
20.	Даны два набора единиц, разделенные *. Построить машину Тьюринга, которая определяет равны ли эти наборы.

3. АНАЛИЗ АЛГОРИТМОВ

3.1 Краткие теоретические сведения

При анализе и разработке машинно-независимых алгоритмов используется **модель обобщенной однопроцессорной машины с произвольным доступом к памяти.** (Random - Access - Machine) или RAM-машиной. В этой модели команды процессора выполняются последовательно; одновременно выполняемые операции отсутствуют.

В рассматриваемой нами RAM – модели, мы можем подсчитать количество шагов, требуемых для исполнения конкретного экземпляра задачи. Но, чтобы понять насколько алгоритм хорош или плох для нас, нужно знать, как он работает со всеми экземплярами задач, то есть на всех входных данных.

Сложность алгоритма в наихудшем случае – это функция, определяемая максимальным количеством шагов, требуемых для обработки любого входного экземпляра размером n.

Сложность работы алгоритма в наилучшем случае – это функция, определяемая минимальным количеством шагов, требуемых для обработки любого входного экземпляра размером n.

Сложность работы алгоритма в среднем случае – это функция, определяемая количеством шагов, требуемых для обработки всех экземпляров размером n.

В практическом смысле наиболее важной является оценка сложности алгоритма в наихудшем случае.

Время работы (running time**) алгоритма** – число элементарных шагов, которые выполняет алгоритм. Время затрачивается также на **вызов** (call) процедуры и ее **исполнение** (execution).

Время работы алгоритма – это сумма промежутков времени, необходимых для выполнения каждой входящей в его состав исполняемой инструкции.

3.1.1 Анализ вычислительной сложности итерационных алгоритмов

В основе итерационных алгоритмов лежит **итерация** – многократное повторение одних и тех же действий. Структура таких алгоритмов описывается алгоритмическими конструкциями «следование», «ветвление», «цикл».

Анализ итерационных алгоритмов сводится к определению трудоемкости этих конструкций и получению интегральной оценки с использованием правил суммы и произведения. Предполагаем, что для выполнения каждой строки псевдокода требуется фиксированное время c_i.

Таким образом время работы алгоритма является функцией от размера входа алгоритма $T(n)$. Нас интересует только главный член получаемой формулы, отражающий **скорость роста** или **порядок роста** функции.

На рисунке 3.1 в псевдокоде представлена итеративная процедура и время выполнения операций в каждой строке.

Greedy_ Activity_Selector(s, f)

1 $n \leftarrow$ length [s]	c_1
2 $A \leftarrow \{ a_1 \}$	c_2
3 $i \leftarrow 1$	c_3

4	**for** $m \leftarrow 2$ **to** n	$c_4 * (n\text{-}1)$
5	**do if** $s_m \geq f_i$	$c_5 *$ (n-2)
6	**then** $A \leftarrow A \cup \{ a_m \}$	$c_6 * (n\text{-}2)$
7	$i \leftarrow m$	$c_7 * (n\text{-}2)$
8	**return** A	c_8

Рисунок 3.1 – Пример итеративной процедуры Greedy_ Activity_Selector(s, f).

Время работы данной процедуры можно оценить (в наихудшем случае) с помощью функции, полученной суммированием времени выполнения инструкций в каждой строке кода:

$T(n) = c_1 + c_2 + c_3 + c_4 * (n\text{-}1) + c_5 * (n\text{-}2) + c_6 * (n\text{-}2) + c_7 * (n\text{-}2) + c_8 = \Theta(n)$

Таким образом время работы процедуры Greedy_ Activity_Selector(s, f), составляющей расписание, описывается полиномом первой степени $\Theta(n)$.

3.1.2. Анализ вычислительной сложности рекурсивных алгоритмов методом деревьев рекурсии и основным методом

Рекурсивный алгоритм – это алгоритм, определяемый через себя. В основе рекурсивных алгоритмов лежит рекурсия, то есть повторение целого в его части.

Пример рекурсивной процедуры на языке Pascal представлен на рисунке 3.2.

```
1        Rec(a: integer);
2          begin
3            if a > 0 then
4                    Rec(a-1);
5            writeln(a);
6          end;
```

Рисунок 3.2 – Пример рекурсивной процедуры Rec.

Процедура Rec вызывает сама себя в четвертой строке представленного кода.

Время работы рекурсивного алгоритма описывается рекуррентным соотношением. При этом требуемое для решения всей задачи с объемом ввода n, выражается через время решения вспомогательных задач. Затем, данное рекуррентное уравнение решается с помощью определенных математических методов, и устанавливаются границы производительности алгоритма.

Например, для алгоритма, основанного на методе «разделяй и властвуй», рекуррентное соотношение будет иметь вид:

$$T(n) = \begin{cases} \Theta(1), & \text{при } n \le c, \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{в противном случае} \end{cases} \qquad (3.1)$$

Если размер задачи достаточно мал ($n \le c$), где c – заранее известная константа, то задача решается непосредственно в течение определенного фиксированного времени, которое мы обозначаем через $\Theta(1)$. В случае, если наша задача делится на a подзадач, объем каждой из которых равен $1/b$ от объема исходной задачи и разбиение задачи на вспомогательные подзадачи происходит в течение времени $D(n)$, а объединение решений подзадач в решение исходной задачи - в течение времени $C(n)$, то мы получим рекуррентное уравнение, расположенное во второй строчке рекуррентного соотношения (3.1).

Для алгоритма сортировки по методу слияния рекуррентное соотношение (3.1) имеет вид:

$$T(n) = \begin{cases} \Theta(1), & \text{при } n = 1, \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{при } n > 1. \end{cases}$$

Существует три метода решения рекуррентных уравнений, то есть получения асимптотических Θ-оценок и O-оценок решения.

Первый метод - **метод подстановок -** заключается в том**,** что мы догадываемся, какой вид имеют граничные функции, а затем, с помощью метода математической индукции доказываем, что догадка была правильной.

Второй метод - **метод деревьев рекурсии** – рекуррентное соотношение преобразуется в дерево, узлы которого представляют время выполнения каждого уровня рекурсии; затем для решения соотношения используется метод оценок сумм.

Третий метод – **основной метод** – граничные оценки решений рекуррентных соотношений представляются в виде:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

где $a \geq 1$, $b > 1$, а функция $f(n)$ - это заданная функция.

Теорема (Основная теорема).

Пусть $a \geq 1$ и $b > 1$ - константы, а $f(n)$ - произвольная функция, а $T(n)$-функция, определенная на множестве неотрицательных целых чисел с помощью рекуррентного соотношения

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

где выражение $\frac{n}{b}$ является целым и интерпретируется либо как верхняя границы отношения $\lceil \frac{n}{4} \rceil$, либо как нижняя граница отношения $\lfloor \frac{n}{4} \rfloor$. Тогда асимптотическое поведение функции можно выразить следующим образом:

1. Если $f(n) = O(n^{\log_b a - \varepsilon})$ для некоторой константы $\varepsilon > 0$, то $T(n) = \Theta(n^{\log_b a})$.
2. Если $f(n) = \Theta(n^{\log_b a})$, то $T(n) = \Theta(n^{\log_b a} \log_2 n)$.
3. Если $f(n) = \Omega(n^{\log_b a + \varepsilon})$ для некоторой константы $\varepsilon > 0$, и если $a \cdot f(n/b) \leq c \cdot f(n)$ для некоторой константы $c < 1$ и всех достаточно больших n, то $T(n) = \Theta(f(n))$.

3.2. Примеры решения заданий

При анализе сложности вложенных циклов мы приходим к вычислению и оцениванию кратных сумм.

Пример 3.2.1 Составить алгоритм для транспонирования (n, n) – матрицы $A = (a_{ij})$ и оценить его вычислительную сложность (время работы).

Решение: Псевдокод итерационного алгоритма для транспонирования (n, n) – матрицы представлен ниже.:

```
1  for i ← 1 to n-1 do
2      for j ← i+1 to n  do
3          a_ij ↔ a_ji
4  return A
```

Число обменов при транспонировании в этом случае, очевидно, будет равно:

$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1}(n-i) = \frac{n^2-n}{2}$. Таким образом $T(n) = \Theta(n^2)$.

Аналогично, чтобы понять, симметрична ли наша матрица потребует от нас в худшем случае $\sum_{i=1}^{n-1}\sum_{j=i}^{n} 1 = \sum_{i=1}^{n-1}(n-i+1) = \frac{(n+1)(n-2)}{2}$ сравнений. И так же $T(n) = \Theta(n^2)$.

В нашем примере суммы нижних границ суммирования не превосходят верхних границ.

Пример 3.2.2 Проведите асимптотический анализ представленного алгоритма:

```
1        s ← 0
2        for i ← 1 to ⌊√(n³ − 1)⌋ do
3            k ← i
4            while k > 1 do
5                s ← s + k;  k ←⌊k/3⌋
```

Решение: Для внутреннего цикла выполняемого при начальном значении k, равном значению i, число шагов и общее число операций допускает оценку $\Theta(\log i)$ – например, представим себе, что число k записано в троичной системе счисления, тогда каждый шаг удаляет одну цифру из записи k.

Затраты на выполнение внешнего цикла представляют собой ряд $\sum_{k=1}^{\varphi(n)} a(k)$, где $\varphi(n) = \lfloor\sqrt{n^3-1}\rfloor$, $a(k) = \Theta(\log k)$. По свойству сумм ряд допускает оценку $\Theta(\sum_{k=1}^{\varphi(n)} \log k)$ или $\Theta(\log(\varphi(n))!)$. Так как $\varphi(n) \to \infty$ при $n \to \infty$, то применив следствие формулы Стирлинга $\Theta(\log(\varphi(n))!) = \Theta(\varphi(n)\log \varphi(n))$, получаем оценку $T(n) = \Theta(\lfloor\sqrt{n^3-1}\rfloor \log \lfloor\sqrt{n^3-1}\rfloor)$. После упрощения, получаем $\Theta(n^{3/2} \log n)$.

Оценку $T(n) = O(n^{3/2} \log n)$ можно было бы получить и не прибегая к формуле Стирлинга. Сумма конечного числа неотрицательных слагаемых не превосходит произведения наибольшего слагаемого на число слагаемых суммы.

Пример 3.2.3 С помощью Основной теоремы найдите асимптотическую оценку для рекуррентного соотношения $T(n) = 6T\left(\frac{n}{3}\right) + 2n$.

Решение: В этом случае $a = 6$, $b = 3$, $f(n) = 2n$, так что

$n^{\log_b a} = n^{\log_3 6} \approx n^{1,63}$

Поскольку $f(n) = 2n = O(n^{\log_3 6 - \varepsilon})$, где $\varepsilon \approx 0{,}63$, то можно применить первый пункт основной теоремы и сделать вывод, что решение имеет вид

$T(n) = \Theta(n^{1,63})$ или $T(n) = O(n^2)$.

Пример 3.2.4 Найти асимптотическую оценку для рекуррентного соотношения $T(n) = 4T\left(\frac{n}{5}\right) + n\log_2 n$.

Решение: В этом случае $a = 4$, $b = 5$, $f(n) = n\log_2 n$, поэтому $n^{\log_b a} = n^{\log_5 4} = O(n^{0,861})$. Так как $f(n) = \Omega(n^{\log_5 4 + \varepsilon})$, где $\varepsilon \approx 0{,}2$, то применим третий пункт основной теоремы. Для этого докажем условие регулярности для функции $f(n)$:

$a \cdot f(n/b) = 4(n/5)\log_2(n/5) \leq (4/5)\, n\log_2 n = c \cdot f(n)$, при $c = 4/5$ и достаточно больших n. Следовательно, согласно пункту 3 Основной теоремы, решение этого рекуррентного соотношения - $T(n) = \Theta(n\log_2 n)$.

Пример 3.2.5 Найти асимптотическую оценку для рекуррентного соотношения $T(n) = 5T\left(\frac{n}{5}\right) + n\log_2 n$

Решение: В этом случае $a = 5$, $b = 5$, а $f(n) = n\log_2 n$ и $n^{\log_b a} = n$. Пункт 3 теоремы мы не можем применить, так как функция $f(n) = n\log_2 n$ не полиномиально больше, чем $n^{\log_b a}$. Отношение $f(n)/ n^{\log_b a} = (n\log_2 n)/ n = \log_2 n$ асимптотически меньше функции n^{ε} для любой константы $\varepsilon > 0$. Следовательно применение Основной теоремы в данном случае невозможно.

Пример 3.2.6 Найти асимптотическую оценку для рекуррентного соотношения $T(n) = T\left(\frac{4n}{5}\right) + 23$.

Решение: В этом случае $a = 1$, $b = 4/5$, $f(n) = 23$, так что $n^{\log_b a} = n^{\log_{5/4} 1} = n^0 = 1$

Поскольку $f(n) = 23$ (константа), то есть $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$, можно применить второй пункт основной теоремы и сделать вывод, что решение - $T(n) = \Theta(\log_2 n)$.

Пример 3.2.7 С помощью метода деревьев рекурсии найти решение рекуррентного уравнения $T(n) = 5T\left(\lfloor \frac{n}{4} \rfloor\right) + 3n^2$.

Решение: На рисунке 3.3*а)* показана функция $T(n)$, которая на рисунках 3.3б) и 3.3в), расписывается в эквивалентное дерево рекурсии, представляющее анализируемое рекуррентное соотношение.

В корне дерева ставим выражение $3n^2$, являющееся временем верхнего уровня рекурсии, а три поддерева, берущих начало из корня, - это время выполнения подзадач размера $n/4$ (рисунок 3.3б).

На рисунке 3.3в) показано дерево рекурсии, которое мы построим, уменьшая размер подзадач и дойдя до граничных условий.

Определим количество уровней, которое нам надо построить, чтобы достичь граничных условий. Размер вспомогательной задачи i–ом уровне глубины равен $n/4^i$. Таким образом размер задачи становится равным единице, когда $n/4^i = 1$, то есть $i = \log_4 n$. Получаем, что с учетом корня, в дереве всего $\log_4 n + 1$ уровней.

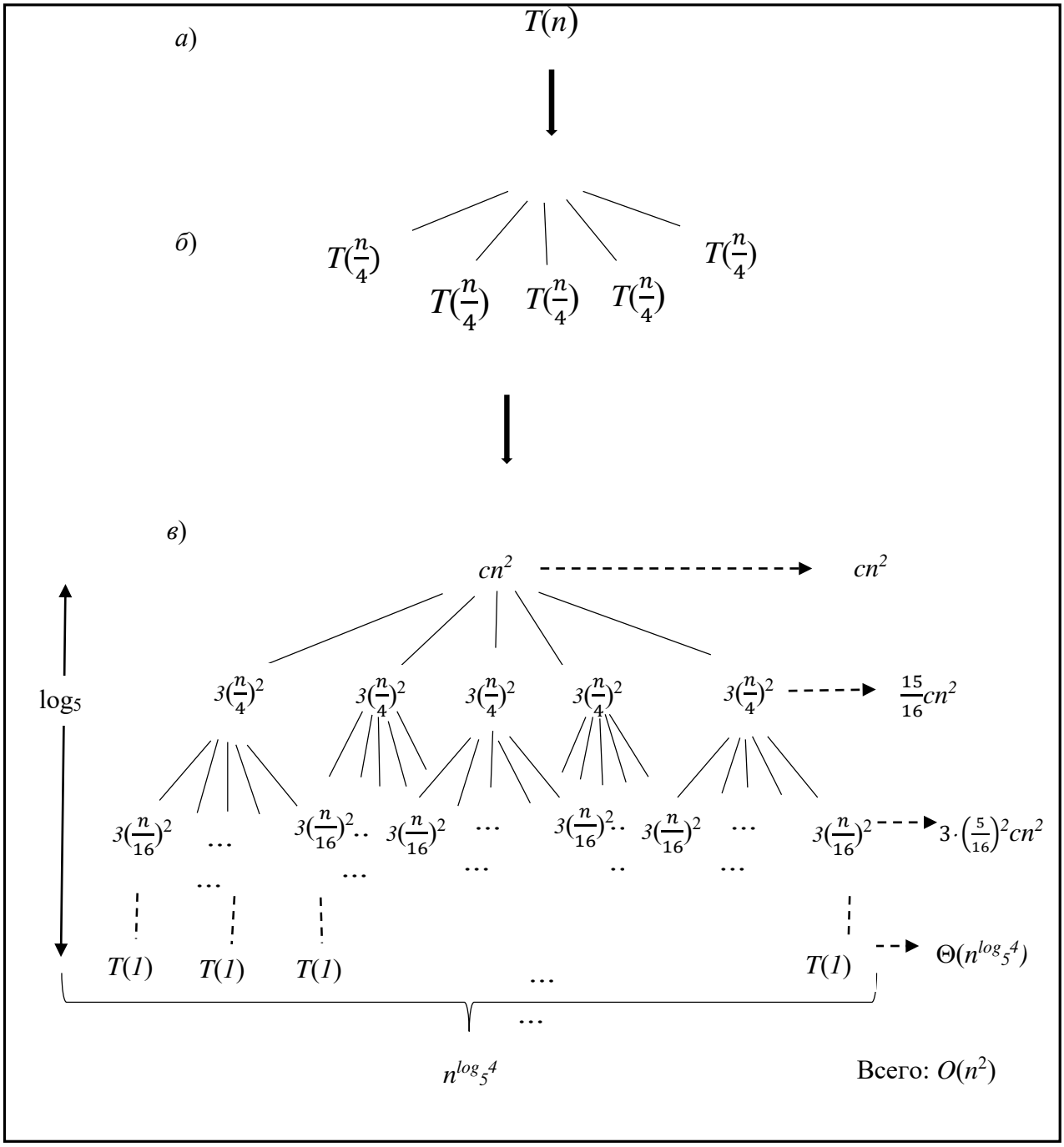


Рисунок 3.3 – Процесс построения дерева рекурсии для рекуррентного соотношения $T(n) = 5T\left(\lfloor \frac{n}{4} \rfloor\right) + 3n^2$.

Теперь определим время выполнения для каждого уровня дерева. На каждом уровне в пять раза больше узлов, чем на предыдущем уровне, поэтому количество узлов на i–ом уровне равно 5^i. Поскольку размеры вспомогательных подзадач при спуске на один уровень уменьшаются в четыре раза, время выполнения каждого узла на i–ом уровне равно $3(n/4^i)^2$. Суммарное время выполнения вспомогательных подзадач на i–ом уровне равно

$$5^i \cdot 3(n/4^i)^2 = 3 \cdot (5/16)^i n^2.$$

Последний уровень, находящийся на глубине $\log_4 n$, состоит из $5^{\log_4 n} = n^{\log_4 5}$ узлов, каждый из которых дает в общее время работы вклад, равный $T(1)$. Поэтому время работы этого уровня равно величине $n^{\log_4 5} T(1)$, которая имеет оценку $\Theta(n^{\log_4 5})$.

Просуммируем время работы по всем уровням дерева и определим время работы алгоритма целиком:

$$T(n) = 3n^2 + \frac{5}{16}3n^2 + \left(\frac{5}{16}\right)^2 3n^2 + \ldots + \left(\frac{5}{16}\right)^{\log_4 n - 1} 3n^2 + \Theta(n^{\log_4 5}) =$$

$$= \sum_{i=0}^{\log_4 n - 1} \left(\frac{5}{16}\right)^i 3n^2 + \Theta(n^{\log_4 5}) = \frac{\left(\frac{5}{16}\right)^{\log_4 n - 1}}{\frac{5}{16} - 1} 3n^2 + \Theta(n^{\log_4 5})$$

Произведем асимптотическую оценку нашей формулы, используя в качестве верхней границы бесконечно убывающую геометрическую прогрессию:

$$\sum_{i=0}^{\log_4 n - 1} \left(\frac{5}{16}\right)^i cn^2 + \Theta(n^{\log_4 5}) < \sum_{i=0}^{\infty} \left(\frac{5}{16}\right)^i 3n^2 + \Theta(n^{\log_4 5}) =$$

$$= \frac{1}{1 - \frac{5}{16}} 3n^2 + \Theta(n^{\log_4 5}) = \frac{16}{11} 3n^2 + \Theta(n^{\log_4 5}) = O(n^2).$$

Из полученного выражения видно, что полное время работы дерева определяется временем работы его корня. Итак, получаем предположительно решение $T(n) = O(n^2)$ – верхняя граница. Эта граница является асимптотической точной оценкой. Действительно, первый рекурсивный вызов дает вклад в общее время работы алгоритма, которое выражается как $\Theta(n^2)$, поэтому нижняя граница решения рекуррентного соотношения представляет собой $\Omega(n^2)$.

3.3 Варианты заданий

Задание 3. Разработайте итеративный алгоритм, опишите его с помощью псевдокода. Составьте блок-схему полученного алгоритма. Реализуйте алгоритм на языке Pascal. Оцените вычислительную сложность алгоритма.

Номер варианта	Задание для реализации
1.	Разработать алгоритм сортировки столбцов двумерного массива целых чисел. Упорядочить элементы столбцов по возрастанию.
2.	Разработать алгоритм сортировки столбцов двумерного массива, состоящего из вещественных чисел. Упорядочить элементы строк по возрастанию
3.	Разработать алгоритм сортировки двумерного массива целых чисел. Упорядочить все элементы массива по возрастанию.
4.	Разработать алгоритм сортировки строк двумерного массива целых чисел. Упорядочить строки массива по убыванию элементов в третьем столбце исходного массива.
5.	Разработать алгоритм сортировки строк двумерного массива вещественных чисел. Упорядочить строки полученного массива по убыванию суммы элементов, входящих в строку исходного массива.
6.	Разработать алгоритм сортировки двумерного массива вещественных чисел. Упорядочить элементы массива по убыванию.
7.	Разработать алгоритм сортировки строк двумерного массива вещественных чисел. Упорядочить строки полученного массива по возрастанию элементов второго столбца.
8.	Разработать алгоритм сортировки элементов двухмерного массива вещественных чисел. Упорядочить элементы полученного массива по убыванию.

9.	Разработать алгоритм сортировки элементов двухмерного массива вещественных чисел. Упорядочить строки полученного массива по убыванию элементов в первом столбце исходного массива.
10.	Разработать алгоритм сортировки элементов строк двумерного массива целых чисел. Упорядочить элементы в строках по возрастанию.
11.	Разработать алгоритм сортировки элементов строк двумерного массива целых чисел. Упорядочить элементы в строках по убыванию.
12.	Разработать алгоритм сортировки элементов строк двумерного массива целых чисел. Упорядочить элементы массива по убыванию.
13.	Разработать алгоритм сортировки столбцов двумерного массива целых чисел. Упорядочить элементы столбцов по возрастанию.
14.	Разработать алгоритм сортировки элементов строк двумерного массива целых чисел. Упорядочить элементы в строках по возрастанию.
15.	Разработать алгоритм сортировки элементов двухмерного массива вещественных чисел. Упорядочить элементы полученного массива по убыванию.
16.	Разработать алгоритм сортировки элементов двухмерного массива вещественных чисел. Упорядочить столбцы полученного массива по убыванию элементов в третьей строке исходного массива.
17.	Разработать алгоритм сортировки столбцов двумерного массива целых чисел. Упорядочить элементы столбцов по возрастанию.

18.	Разработать алгоритм сортировки столбцов двумерного массива, состоящего из символов. Упорядочить элементы столбцов по возрастанию.
19.	Разработать алгоритм сортировки столбцов двумерного массива целых чисел. Упорядочить элементы столбцов по убыванию.
20.	Разработать алгоритм сортировки строк двумерного массива целых чисел. Упорядочить элементы строк по возрастанию.

Задание 4. Пронумеруйте строки в итеративном алгоритме, начиная с первой, и определите, прописав сумму ряда, какое количество раз повторяется *tt*-я и *qq*-я строки в приведенных программах (фрагментах программ).

Номер варианта	Номера строк

Для программы soch:

1.	*tt* = 7, *qq* = 27;
2.	*tt* = 9, *qq* = 22;
3.	*tt* = 13, *qq* = 20;
4.	*tt* = 11, *qq* = 23;

```
program soch;
const n = 5; r = 3;
var i,j,l,c,f: integer;
m:array[1..n] of integer;
begin
c:=r+1;
for i:=r+2 to n do c:=c*i;
f:=1;
for i:=2 to n-r do f:=f*i;
c:= c div f;
```

```
for i:=1 to n do
 begin
  m[i]:=i;
  if i<=r then write(m[i]:1);
 end;
for i:=2 to c do
 begin
  f:=0; j:=r;
  repeat
   if m[j]< (n-r+j) then
    begin
     m[j]:=m[j]+1;
     for l:=j+1 to r do
      m[l]:= m[j]+l-j;
     f:=1;
    end;
   j:=j-1;
  until f=1;
  for j:=1 to r do write(m[j]:1);
  write(' ');
 end;
writeln;
end.
```

Для процедуры summa:

5.	*tt* = 7, *qq* = 16;

```
procedure summa(var a, b: array [1..20]  of  longint; n, m: integer);
var
  s: longint;
  i: integer;
begin
```

```
  s := a[1];
  for i := 1 to n - 1 do
    s := s + (2 * b[i] - 1) * a[i + 1];
  if s = m then
  begin
    write(a[1]);
    for i := 1 to n - 1 do
    begin
      if b[i] = 1 then write('+')
                  else write('-');
     write(a[i + 1]);
    end;
    writeln(' = ', s:4);
  end;
end;
```

Для программы raz:

6.	*tt* = 9, *qq* = 13;
7.	*tt* = 7, *qq* = 11;

```
program raz;
const n = 5; r = 3;
var i,j,a,k: integer;
b:array[1..n] of integer;
begin
 a:=n;
 for i:= 1 to n do read(b[i]);
 readln;
for i:=1 to a do
 begin
  k:=i-1;
  for j:=1 to r do
```

```
  begin
   write(b[(k mod n)+1]:1); k:=k div n;
  end;
 write(‘ ‘);
writeln;
end.
```

	Для программы razp:
8.	*tt* = 8, *qq* = 11;
9.	*tt* = 7, *qq* = 13;
10.	*tt* = 11, *qq* = 21;
11.	*tt* = 10, *qq* = 23;
12.	*tt* = 6, *qq* = 26;

```
program razp;
const n = 5; r = 3;
var i,j,a,k, s,p: integer;
b:array[1..n] of integer;
begin
 a:=n;
for i:=2 to r do a:=a*n;
for i:=1 to a do
 begin
  k:= i-1; s:=0;
  for j:=1 to r do
   begin
    s:=s+(k mod n) +1);
    k: = k div n;
   end;
  if s=8 then
   begin
```

```
    k:=i-1;
    for j:=1 to r do
     begin
      write(((k mod n) +1); k:=k div n;
     end;
    write(‘ ‘);
   end;
 end;
writeln;
end.
```

Для программы razpp:

13.	*tt* = 6, *qq* = 12;
14.	*tt* = 7, *qq* = 15;
15.	*tt* = 9, *qq* = 13;

```
program razpp;
const n = 5; r = 3;
var i,j,a,k, s,p: integer;
begin
 p:=0; a:=n;
 for i:=2 to r do a:=a*n;
 for i:=1 to a do
  begin
   k:= i-1; s:=0;
    for j:=1 to r do
     begin
      s:=s+(k mod n) +1;
      k:=k div n;
     end;
    if s=8 then
     p:=p+1;
```

```
 end;
 writeln('p=',p:8);
end.
```

Для процедуры backtrack:

16.	*tt* = 10, *qq* = 18;
17.	*tt* = 5, *qq* = 32;
18	*tt* = 12, *qq* = 27;
19.	*tt* = 14, *qq* = 31;
20.	*tt* = 6, *qq* = 16.

```
procedure backtrack( var a: array [1..20]  of  longint; n: integer);
const  m: longint = 0;
var  s: array [1..20]  of  longint; p, i, k: integer;
begin
  p := 0;
  for i := 1 to n do  s[i] := 0;
  s[1] := 1;  k := 1;
  while k > 0 do
   begin
    while s[k] <= n do
     begin
      a[k] := s[k];
      repeat
        s[k] := s[k] + 1;
      until (s[k] > n) or flag(a, s[k], k);
      if k = n then
       begin
        p := p + 1;
        write(p, ' ');
        for i := 1 to n do
```

```
        write( a[i], ' '); writeln;
      end {if}
    else
    begin
      k := k + 1;
      s[k] := 1;
      while (s[k] <= n) and not flag(a, s[k], k) do
        s[k] := s[k] + 1;
    end{else}
  end;{while}
  k := k - 1;
 end;{while}
end;{backtrack}
```

Задание 5. Решите рекуррентное соотношение любым известным вам методом.

Номер варианта	Рекуррентное соотношение
1.	$T(1)=1, T(n)=2T(n/2)+6n-1, \forall n \geq 2.$
2.	$T(1)=4, T(n)=2T(n/2)+3n+2, \forall n \geq 2.$
3.	$T(1)=1, T(n)=6T(n/6)+2n+3, \forall n \geq 2.$
4.	$T(1)=3, T(n)=6T(n/6)+3n-1, \forall n \geq 2.$
5.	$T(1)=2, T(n)=4T(n/3)+3n-5, \forall n \geq 2.$
6.	$T(1)=1, T(n)=3T(n/2)+n^2-n, \forall n \geq 2.$
7.	$T(1)=4, T(n)=3T(n/2)+n^2-2n+1, \forall n \geq 2.$
8.	$T(1)=1, T(n)=3T(n/2)+n-2, \forall n \geq 2.$
9.	$T(1)=1, T(n)=3T(n/2)+5n-7, \forall n \geq 2.$
10.	$T(1)=1, T(n)=4T(n/3)+n^2, \forall n \geq 2.$

11.	$T(1)=1,\ T(n)=3T\left(\frac{n}{4}\right)+n^2,\ \forall\ n\geq 2.$
12.	$T(n)=3T\left(\frac{n}{4}\right)+n^2$
13.	$T(n)=3T\left(\frac{n}{4}\right)+log_2 n,$
14.	$T(n)=5T\left(\frac{n}{4}\right)+n\cdot log_2 n$
15.	$T(n)=5T\left(\frac{n}{4}\right)+n$
16.	$T(n)=9T\left(\frac{n}{3}\right)+n$
17.	$T(n)=3T\left(\frac{n}{9}\right)+n^2$
18.	$T(n)=9T\left(\frac{n}{9}\right)+n^2$
19.	$T(1)=1,\ T(n)=3T\left(\frac{n}{2}\right)+n^2,\ \forall\ n\geq 2.$
20.	$T(n)=4T\left(\frac{n}{4}\right)+n^2+1$

4. НЕКОТОРЫЕ МЕТОДЫ ПОДСЧЕТА И ОЦЕНИВАНИЯ В ДИСКРЕТНОЙ МАТЕМАТИКЕ

4.1 Краткая теоретические сведения

Существуют различные аналитические методы для анализа последовательностей. В том числе производящие функции и методы исчисления конечных сумм. Рассматриваемые методы необходимы при решении комбинаторных задач дискретной математики, где объектом исследования часто являются последовательности. Так же для алгоритма, содержащего итеративную управляющую конструкцию, время работы алгоритма можно представить в виде суммы значений времени выполнения отдельных итераций. В результате мы решаем задачу вычисления суммы ряда, получая асимптотику для времени работы алгоритма.

4.1.1 Производящие функции

Последовательности можно сопоставить функцию действительного или комплексного переменного, тогда последовательность будет рассматриваться не как набор чисел, а как единое целое.

Поставим в соответствие последовательности бесконечный ряд $\sum_{k=0}^{\infty} a_k z^k$. При этом мы не ограничиваем общности, так как любую конечную последовательность можно представить в виде бесконечной, в которой все ее члены, начиная с некоторого номера, равны нулю. Сумма этого ряда и называется производящей функцией для данной последовательности:

$G(z) = \sum_{k=0}^{\infty} a_k z^k$, где z – комплексное число.

Из теории функции комплексного переменного рассматриваемый степенной ряд абсолютно сходится в некотором круге комплексной плоскости с центром в точке $z = 0$. Члены последовательности $(a_1, a_2, a_3, \ldots, a_n, \ldots)$ являются коэффициентами ряда Тейлора в этой точке $z = 0$.

Из математического анализа для вычисления коэффициентов ряда Тейлора используется формула

$a_n = \frac{1}{n!} D^n (G(z))\big|_{z=0}$, где через $D^n (G(z))\big|_{z=0}$ обозначается производная порядка n функции $(G(z))$ в точке $z = 0$.

Существует связь между некоторыми конкретными последовательности и производящими функциями (см. пример п.4.2.2).

Производящие функции обладают свойством линейности по отношению к порождающим их последовательностям, а именно, если у нас имеется две последовательности $(a_1, a_2, a_3, \ldots, a_n, \ldots)$ и $(b_1, b_2, b_3, \ldots, b_n, \ldots)$, производящие функции которых равны соответственно $G_1(z)$ и $G_2(z)$, то для последовательности $(c \cdot a_0 + d \cdot b_0, c \cdot a_1 + d \cdot b_1, \ldots, c \cdot a_n + d \cdot b_n, \ldots)$ производящая функция равна:

$G(z) = c \cdot G_1(z) + d \cdot G_2(z)$.

Пусть даны две последовательности $A = (a_0, a_1, a_2, \ldots, a_n, \ldots)$ и $B = (b_0, b_1, b_2, \ldots, b_n, \ldots)$, производящие функции которых равны $G_1(z)$ и $G_2(z)$ соответственно. Тогда сверткой $C = A*B$ называется последовательность $C = (c_0, c_1, c_2, \ldots, c_n, \ldots)$, такая что $c_n = \sum_{k=0}^{\infty} a_k\, b_{n-k}$.

Рассмотрим производящую функция $G(z) = (z + 1)^n$ соответствующую последовательности $(a_0, a_1, a_2, \ldots, a_n, \ldots)$ и являющуюся производящей для числа сочетаний без повторений из n по k (см. пример 4.2.1). Если у нас есть производящие функции $G_1(z) = (z + 1)^m$ с соответствующей последовательностью $(b_1, b_2, b_3, \ldots, b_n, \ldots)$ и $G_2(z) = (z + 1)^r$ с соответствующей последовательностью $(c_1, c_2, c_3, \ldots, c_n, \ldots)$, и $m + r = n$, то произведению этих функций соответствует функция

$$G(z) = G_1(z)\cdot G_2(z) = (z + 1)^m \cdot (z + 1)^r = (z + 1)^{m+r} = (z + 1)^n .$$

Этой производящей функции соответствует последовательность $(a_1, a_2, a_3, \ldots, a_k, \ldots)$, где $a_k = C_n^k$. С другой стороны, последовательность $(a_1, a_2, a_3, \ldots, a_k, \ldots)$ является сверткой последовательностей $(b_1, b_2, b_3, \ldots, b_k, \ldots)$ и $(c_1, c_2, c_3, \ldots, c_k, \ldots)$. Таким образом для любого k имеем $a_k = \sum_{l=0}^{k} b_l\, c_{k-l}$. Получаем

$$C_n^k = \sum_{l=0}^{k} C_m^l\, C_r^{k-1}.$$

Так как $m + r = n$, то $C_{r+m}^k = \sum_{l=0}^{k} C_m^l\, C_r^{k-1}$. Это соотношение называется ***правилом свертки Вандермонда***.

4.1.2 Некоторые методы исчисление конечных сумм

Исчисление конечных сумм является одним из важнейших разделов дискретной математики, применяемом при анализе алгоритмов для получения явных функциональных зависимостей от размера входа алгоритма.

Например, рассмотрим приведенный ниже фрагмент программы:

```
for i = 1 to n - 1
  for j = i to n - i +1
     <тело цикла>
  end /for j
```

end /for *i*

Сколько раз будет выполнено тело цикла, если $n = 11$? Ответ можно получить вычислительным экспериментом. Но чтобы получить функциональную зависимость количества выполнений тела цикла от n, потребуется применение аппарата исчисления конечных сумм. Математически задача сводится к получению функции

$S(n) = n + (n - 2) + (n - 4) + \ldots$

В общем виде конечная сумма имеет вид:

$a_1+a_2 + \ldots + a_n$, где a_k - слагаемое суммы с индексом k.

При этом предполагаем, что $a_k = f(k)$, то есть вычисляется как некоторая заданная функция аргумента k. В сигма обозначениях сумма имеет вид

$$S(n) = \sum_{k=1}^{n} a_k.$$

При оценке сумм нас, как правило, в задачах анализа алгоритмов интересуют оценки при достаточно больших n. При этом мы предполагаем, что функции неотрицательные и стремятся к бесконечности при достаточно больших аргументах.

Пусть $g(n)$ – некоторая функция. Тогда запись $\boldsymbol{\Theta(g(n))}$ –обозначает множество функций $\Theta(g(n)) = f(n)$ таких что

$$\exists\, c_1 , c_2 > 0 \text{ и } \exists\, n_0 : 0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n) \ \forall n \ge n_0$$

Функция *g(n)* называется ***асимптотически точной оценкой функции f(n).***

Пусть $g(n)$ – некоторая функция, тогда запись $\boldsymbol{O(g(n))}$ –обозначает множество функций $O(g(n)) = f(n)$ таких что

$$\exists\, c > 0 \text{ и } \exists\, n_0 : \quad 0 \le f(n) \le c \cdot g(n) \ \forall n \ge n_0$$

Функция *g*(*n*) называется **асимптотической верхней границей функции *f*(*n*)**.

Пусть $g(n)$ – некоторая функция. Тогда запись $\boldsymbol{\Omega\,(g(n))}$ –обозначает множество функций $\Omega(g(n)) = f(n)$ таких что

$$\exists\, \boldsymbol{c} > \mathbf{0} \text{ и } \exists\, \boldsymbol{n_0} : 0 \le c \cdot g(n) \le f(n) \ \forall n \ge n_0$$

Функция g(n) называется ***асимптотической нижней границей функции f*(*n*).**

При оценке сумм возможно применение ***метода математической индукции***, когда мы «угадываем» решение, проверяем его правильность при n = 1, и, считая, что результат верен при $n = k$, доказываем его для $n = k + 1$.

Метод почленных сравнений основан на свойствах линейности и коммутативности членов конечных сумм. Пусть имеются две конечные суммы

$$S_1(n) = \sum_{k=1}^{n} a_k,\ S_2(n) = \sum_{k=1}^{n} b_k$$

и для всех значений выполняется условие $a_k \le b_k$. Тогда можно утверждать, что $S_1(n) \le S_2(n)$.

Метод сравнения с определенным интегралом удобно применять для исследования сумм, слагаемые которых имеют вид $a_k = f(k)$. И при этом функция действительной переменной $y = f(k)$ неотрицательна и монотонна для положительных значений переменной x.

Если функция не убывает, то, очевидно,

$$a_k \le \int_k^{k+1} f(x)dx \text{ и } a_k \ge \int_{k-1}^{k} f(x)dx.$$

Тогда для значения суммы-функции $S(n) = \sum_{k=1}^{n} a_k$ имеем

$$S(n) \le \int_1^{n+1} f(x)dx \text{ и } S(n) \ge a_1 + \int_1^{n} f(x)dx.$$

Если функция $y = f(x)$ не возрастает, то

$$a_k \ge \int_k^{k+1} f(x)dx \text{ и } a_k \le \int_{k-1}^{k} f(x)dx.$$

В этом случае для функции $S(n)$ имеем

$$S(n) \ge \int_1^{n+1} f(x)dx \text{ и } S(n) \le a_1 + \int_1^{n} f(x)dx.$$

Метод неопределенных коэффициентов основан на том, что вид функции $S(n)$ известен, например, на основе полученных асимптотических оценок, а конкретные значения коэффициентов могут быть получены с использованием метода математической индукции.

Метод прибавления нуля использует специальный прием доказательств, а именно прибавления нуля, записанного в виде некоторого выражения равного нулю. Наряду с этим используется также тождественного переписывание выражений другим образом и умножение на единицу, которая записывается так же в виде некоторого выражения равного единицы.

4.2. Примеры решения заданий

Пример 4.2.1 Найти последовательность, соответствующую производящей функции $G(z) = (az + b)^n$.

Решение: Соответствующие производные имеют вид, представленый ниже.

$D((az + b)^n) = n(az + b)^{n-1}a$;

$D^2((az + b)^n) = n(n-1)(az + b)^{n-2}a^2$;

$D^k((az + b)^n) = n(n-1)\ldots(n - k + 1)(az + b)^{n-k}a^k$, при $k \le n$;

$D^k((az + b)^n) = 0$, при $k > n$.

Применяя формулу Тейлора, получаем:

$$\begin{cases} a_0 = b^n n;\ a_1 = nab^{n-1}; a2 = \frac{1}{2}n(n-1)a^2b^{n-1}; \\ a_k = n(n-1)\ldots(n-k+1)a^k b^{n-k}, k \le n; \\ a_k = 0, k > n. \end{cases}$$

Заметим, что $\frac{1}{k!}n(n-1)\ldots(n-k+1) = C_n^k$.

Таким образом данной производящей функции соответствует последовательность $(a_1, a_2, a_3, \ldots, a_n, \ldots)$, где

$$\begin{cases} a_k = C_n^k a^k b^{n-k}, k \le n; \\ a_k = 0, k > n. \end{cases}$$

Рассмотрим частный случай этой формулы: $G(z) = (z - b)^n$. Соответствующая последовательность будет иметь вид:

$$\begin{cases} a_k = C_n^k (-1)^{n-k} b^{n-k}, k \le n; \\ a_k = 0, k > n. \end{cases}$$

В случае, когда $G(z) = (z + 1)^n$, последовательность задается следующим образом:

$$\begin{cases} a_k = C_n^k; \\ a_k = 0, k > n. \end{cases}$$

То есть функция $G(z) = (z + 1)^n$ является производящей для числа сочетаний без повторений из n по k.

Пример 4.2.2 Найти последовательность, соответствующую производящей функции $G(z) = \frac{1}{(az + b)^{m+1}}$, $m \ge 0$.

Решение:

$D^k(\frac{1}{(az+b)^{m+1}},) = (-1)^k \frac{(m+1)(m+2)\dots(m+k)}{b^{m+1+k}} a^k$,

Тогда

$$a_k = \frac{1}{k!}(-1)^k \frac{(m+1)(m+2)\dots(m+k)}{b^{m+1+k}} a^k = (-1)^k \frac{a^k}{b^{m+1+k}} \frac{1}{k!} \frac{1\cdot 2\cdots(m+k)}{1\cdot 2\cdots m} = (-1)^k \frac{a^k}{b^{m+1+k}} C_{m+k}^m.$$

В частном случае, когда $G(z) = \frac{1}{(z+1)^{m+1}}$ соответствующая последовательность имеет вид $a_k = (-1)^k C_{m+k}^m$.

Пример 4.2.3 Используя производящую функцию найти формулу для выражения *n* – го члена ряда Фибоначчи.

Решение: Рекуррентное соотношение для ряда Фибоначчи имеет вид:

$$\begin{cases} a_0 = 0; \\ a_1 = 1; \\ a_n = a_{n-1} + a_{n-2}, \quad n \geq 2. \end{cases}$$

Введем производящую функцию $G(z) = \sum_{n=0}^{\infty} a_n z^n$, умножим обе части рекуррентного соотношения на z^n и просуммируем по *n*, так что $2 \leq n < \infty$:

$\sum_{n=2}^{\infty} a_n z^n = \sum_{n=2}^{\infty} a_{n-1} z^n + \sum_{n=2}^{\infty} a_{n-2} z^n$.

В то же время

$\sum_{n=2}^{\infty} a_n z^n = \sum_{n=0}^{\infty} a_n z^n - a_0 - a_1 z = G(z) - z$,

$\sum_{n=2}^{\infty} a_{n-1} z^n = \sum_{n=1}^{\infty} a_n z^{n+1} = \sum_{n=0}^{\infty} a_n z^{n+1} - a_0 z = z \sum_{n=0}^{\infty} a_n z^n = zG(z)$,

$\sum_{n=2}^{\infty} a_{n-2} z^n = \sum_{n=0}^{\infty} a_n z^{n+2} = z^2 G(z)$.

На основании этого получаем уравнение для производящей функции

$G(z) - z = zG(z) + z^2 G(z)$,

$G(z) = \frac{-z}{z^2+z-1} = \frac{-z}{(z-\frac{-1+\sqrt{5}}{2})(z-\frac{-1-\sqrt{5}}{2})} = \frac{1}{\sqrt{5}} \cdot (-\frac{\frac{-1+\sqrt{5}}{2}}{z-\frac{-1+\sqrt{5}}{2}} + \frac{\frac{-1-\sqrt{5}}{2}}{z-\frac{-1-\sqrt{5}}{2}})$, тогда

$$a_n = \frac{1}{\sqrt{5}} (\frac{2^n}{(-1+\sqrt{5})^n} - \frac{2^n}{(-1-\sqrt{5})^n}) = \frac{1}{\sqrt{5}} (\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}} (\frac{1-\sqrt{5}}{2})^n .$$

Пример 4.2.4 Сколько существует способов выдать сумму в 12 копеек, используя монеты достоинством в 1 и 5 копеек.

Решение: Число способов выдать сумму в k копеек однокопеечными монетами образует последовательность (b_0, b_1, b_2, …, b_k, …), где b_k,= 1. Производящая функция этой последовательности имеет вид:

$$G_1(z) = 1 + z + z^2 + \ldots z^k + \ldots = \frac{1}{1-z}\,.$$

Для монет достоинством 5 копеек соответствует последовательность (c_0, c_1, c_2, …, c_n, …), где c_k= 1, если k кратно 5 и c_k= 0 в противном случае. Производящая функция этой последовательности имеет вид:

$$G_2(z) = 1 + z^5 + \ldots z^{5k} + \ldots = \frac{1}{1-z^5}\,.$$

Тогда последовательность числа способов для сумм, которые можно выдать однокопеечными и пятикопеечными монетами является сверткой данных последовательностей, и, значит, ее производящая функция

$$G(z) = G_1(z)\cdot G_2(z) = \frac{1}{1-z}\cdot\frac{1}{1-z^5}\,.$$

Тогда число способов, которыми можно выдать сумму в n копеек равно коэффициенту при z^n. Найдем этот коэффициент при n = 12.

Для нахождения коэффициента воспользуемся представлением функций в виде ряда. Заметим, что z^{12} можно представить тремя способами:

$z^{12} = z^{12}\cdot 1$, $z^{12} = z^7\cdot z^5$, $z^{12} = z^2\cdot z^{10}$.

Следовательно, всего существует три способа выдать сумму в 12 копеек, используя монеты достоинством в 1 и 5 копеек.

Пример 4.2.5 Получить оценку O – большое для суммы $S(n) = \sum_{k=1}^{n} 3k^3$.

Решение: Число три, как константу, можно вынести за знак суммы. Далее, очевидно, что для всех $1 \le k \le n$ справедливо равенство $k^3 \le n^3$, и

$$S(n) = 3\sum_{k=1}^{n} k^3 \le 3\sum_{k=1}^{n} n^3 = 3n^4.$$

Отсюда, $S(n) = O(n^4)$.

Пример 4.2.6 Используя метод математической индукции, докажите, что функция $g(n) = n^4$ является асимптотически точной оценкой для суммы $S(n) = \sum_{k=1}^{n} k^3$.

Решение: Для доказательства в соответствии с определением оценки Θ требуется найти такое значение константы c_1 и такое значение n, что при всех $n > n_0$ будет выполняться

$$S(n) = \sum_{k=1}^{n} k^3 \geq c_1 n^4.$$

Требуется так подобрать c_1, чтобы для всех $n > n_0$ было выполнено неравенство $S(n+1) = \sum_{k=1}^{n+1} k^3 \geq c_1(n+1)^4$.

Имеем, $S(n+1) = \sum_{k=1}^{n+1} k^3 + (n+1)^3 = S(n) + (n+1)^3 \geq c_1 n^4 + (n+1)^3$.

Подберем значение c_1, чтобы выполнялось $c_1 n^4 + (n+1)^3 \geq c_1(n+1)^4$.

Таким образом, $c_1 \leq \frac{(n+1)^3}{4n^3+6n^2+4n+1}$.

Анализ неравенства позволяет говорить о том, что при всех $n \geq 1$

$\frac{(n+1)^3}{4n^3+6n^2+4n+1} > \frac{n^3}{4n^3+6n^3+4n^3+n^3} = \frac{1}{15}$, то есть можно выбрать $c_1 = \frac{1}{15}$.

Очевидно, $S(1) = 1^3 \geq \frac{1}{15} 1^4$.

Следовательно, мы можем утверждать, что при всех $n \geq 1$ выполняется

$$S(n) = \sum_{k=1}^{n} k^3 \geq \frac{1}{15} n^4.$$

Ранее мы показали, что при всех $n \geq 1$ верно, что $S(n) \leq n^4$, следовательно при значениях $c_1 = \frac{1}{15}$, $c_2 = 1$, $S(n) = \Theta(n^4)$ и функция $g(n) = n^4$ является асимптотически точной оценкой для функции $S(n)$.

Пример 4.2.7 Используя метод прибавления нуля вычислите сумму геометрической прогрессии.

Решение: Пусть имеется сумма $S(n) = \sum_{k=1}^{n} aq^{k-1}$.

Преобразуем сумму к виду

$$S(n) = \sum_{k=1}^{n} aq^{k-1} = a + \sum_{k=2}^{n} aq^{k-1} = a + q \sum_{k=2}^{n} aq^{k-2} = a +$$

$$+ (q \sum_{k=2}^{n} aq^{k-2} + aq^n) - aq^n,$$

мы прибавили ноль, записанный в виде $0 = aq^n - aq^n$, далее получаем

$$S(n) = a + (q \sum_{k=2}^{n} aq^{k-2} + aq^n) - aq^n = a + q(\sum_{k=2}^{n} aq^{k-2} + aq^{n-1}) - aq^n =$$

$$= a + q(\sum_{k=2}^{n+1} aq^{k-2}) - aq^n = a(1 - q^n) + q(\sum_{k=1}^{n} aq^{k-1}).$$

Отсюда, так как $S(n) = \sum_{k=1}^{n} aq^{k-1}$, то $S(n) = a(1 - q^n) + S(n)$.

И получаем известную нам формулу для суммы ряда геометрической прогрессии $S(n) = a\frac{q^n-1}{q-1}$.

4.3. Варианты заданий

Задание 6. Найти последовательности, которые соответствуют следующим производящим функциям.

Номер варианта	Функция
1, 14.	$G(z) = \frac{1}{1+z}$;
2, 15.	$G(z) = \frac{1}{1-z}$;
3, 16.	$G(z) = \frac{1}{(z-b)^{m+1}}$;
4, 17.	$G(z) = \frac{1}{(z+1)^{m+1}}$;
5, 18.	$G(z) = \frac{1-z^{n+1}}{z-1}$;
6, 19.	$G(z) = \frac{1-z^{n+1}}{1-z}$;
7, 20.	$G(z) = \frac{3}{1-z}$;
8, 11.	$G(z) = \frac{1-z^{n+1}}{1-z^n}$;
9, 13.	$G(z) = \frac{1}{(z-1)^2(2z+1)}$;
10, 12.	$G(z) = \frac{z}{z^2-6z+8}$.

Задание 7. Имеется множество чисел $A = \{a1, \ldots, a_n\}$ Сколькими способами, суммируя эти числа, можно получить число b. Порядок суммирования не важен.

Номер варианта	Множество A и число b.
1.	$A = \{2, 3, 4\}$, $b = 12$;
2.	$A = \{1, 3, 4\}$, $b = 12$;

3.	$A = \{1, 2, 3, 4, 5\}$, $b = 6$;
4.	$A = \{1, 3, 4\}$, $b = 7$;
5.	$A = \{1, 2, 3\}$ $b = 5$;
6.	$A = \{1, 3, 4\}$, $b = 12$
7.	$A = \{1, 2, 4\}$, $b = 8$;
8.	$A = \{1, 3\}$, $b = 9$;
9.	$A = \{2, 3, 4\}$, $b = 7$;
10.	$A = \{1, 2, 3\}$, $b = 10$;
11.	$A = \{1, 3, 4\}$, $b = 11$;
12.	$A = \{1, 4, 5\}$, $b = 15$;
13.	$A = \{1, 3, 4\}$, $b = 8$;
14.	$A = \{1, 3, 5\}$, $b = 10$;
15.	$A = \{1, 7\}$, $b = 15$;
16.	$A = \{2, 5\}$, $b = 12$;
17.	$A = \{1, 4\}$, $b = 11$;
18.	$A = \{2, 4\}$, $b = 12$;
19.	$A = \{4, 6\}$, $b = 12$;
20.	$A = \{3, 4\}$, $b = 15$.

Задание 8.

Номер варианта	Задание
1.	Докажите методом математической индукции формулы для нахождения суммы арифметической прогрессии
2.	Используя метод неопределенных коэффициентов найдите точную сумму $S(n) = \sum_{k=1}^{n} k^3$.
3.	Используя метод добавления нуля, найдите функциональную зависимость для суммы $S(n) = \sum_{k=1}^{n} \frac{1}{k(k+2)} = \sum_{k=1}^{n} (\frac{1}{2k} - \frac{1}{2(k+2)})$.

4.	Используя метод сравнения с определенным интегралом, найти эквивалентную оценку для суммы $S(n) = \sum_{k=1}^{n} k^3$.
5.	Докажите методом сравнения с определенным интегралом, что $\sum_{k=1}^{n} \frac{1}{k^2} = O(1)$.
6.	Используйте метод сравнения с определенным интегралом для исследования суммы $S(n) = \sum_{k=1}^{n} \ln k$.
7.	Используя метод почленных сравнений получите асимптотическую оценку для суммы $S(n) = \sum_{k=1}^{n} \frac{k}{k+1} 2^{k-1}$.
8.	Найдите точную асимптотическую оценку для суммы $S(n) = \sum_{k=1}^{n} \sqrt{k}$.
9.	Найдите точную асимптотическую оценку для суммы $S(n) = \sum_{k=1}^{n} 1/\sqrt{k}$.
10.	Найдите оценку для суммы $S(n) = \sum_{k=1}^{n} \frac{1}{\sqrt{k^3}}$.
11.	Дайте оценку сумме $S(n) = \sum_{k=1}^{n} (2n - 2k)$
12.	Методом определенных интегралов найдите оценку для суммы $S(n) = \sum_{k=1}^{n} \frac{1}{k}$.
13.	Используя метод неопределенных коэффициентов найдите точную сумму $S(n) = \sum_{k=1}^{n} k^5$.
14.	Найдите оценку для суммы $S(n) = \sum_{k=1}^{n} \frac{1}{k^4}$.
15.	Найдите точную асимптотическую оценку для суммы $S(n) = \sum_{k=1}^{n} 2\sqrt{k}$.
16.	Используя метод почленных сравнений получите асимптотическую оценку для суммы $S(n) = \sum_{k=1}^{n} \frac{k}{2k+1} 2^{2k-1}$
17.	Используя метод сравнения с определенным интегралом, найти эквивалентную оценку для суммы $S(n) = \sum_{k=1}^{n} k^6$.
18.	Найдите оценку для суммы $S(n) = \sum_{k=3}^{n} \frac{1}{n^2-4} 2^{k-1}$.

19.	Используя метод неопределенных коэффициентов найдите точную сумму $S(n) = \sum_{k=1}^{n} k^5$
20.	Используя метод почленных сравнений получите асимптотическую оценку для суммы $S(n) = \sum_{k=1}^{n} \frac{k}{2k+2} 2^{k-1}$.

5. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

№	Автор	Наименование, издательство, год издания
1.	Иванов Б.Н.	Дискретная математика. Алгоритмы и программы. Полный курс.-М.ФИЗМАТЛИТ. 2007. – 408 с.
2.	Хаггарти Р.	Дискретная математика для программистов. – 2-е, дополненное издание: Пер. с англ.- М.: ТЕХНОСФЕРА, 2005. – 400 с.
3.	Матрос Д.Ш., Поднебесова Г.Б.	Теория алгоритмов: учебник. -М.: БИНОМ. Лаборатория знаний, 2008. – 202 с.
4.	Новиков Ф.А.	Алгоритмы дискретной математики для программистов. – СПб.: Питер, 2001.
5.	Кнут Д.Э.	Искусство программирования. Том 4, А. Комбинаторные алгоритмы. Часть 1. – Издательство: Вильямс, 2012, с. 960.
6.	Куликов В.В.	Дискретная математика: Учебное пособие. – М.: Риор, 2018.- 448 с.
7.	Иванов А.А., Пронина Г.И., Корягина Н.Ю.	Дискретная математика для инженера: Учебник – СПб.: Лань П, 2016. – 400 с.